



# PROGRAMACIÓN II - UNIDAD 2

# Agenda

- ❖ Desarrollo y utilización de clases y objetos. Variables de clases e instancias.
- ❖ Constructores y destructores.
- ❖ Encapsulamiento.
- ❖ Herencia simple y herencia múltiple.
- ❖ Polimorfismo.
- ❖ Clases de “nuevo-estilo”.
- ❖ Métodos especiales.
- ❖ Tipos en Python.

# Desarrollo y utilización de clases y objetos.

## *Clases / Objetos de Python*

Python es un lenguaje de programación orientado a objetos.

Casi todo en Python es un objeto, con sus propiedades y métodos.

Una clase es como un constructor de objetos o un "blueprint" para crear objetos.

### *Cómo crear una clase:*

Para crear una clase, use la palabra clave class

```
class MyClass:  
    x = 5  
  
print(MyClass)
```

```
<class '__main__.MyClass'>
```

# Desarrollo y utilización de clases y objetos.

## ***Crear objeto***

Ahora podemos usar la clase llamada MyClass para crear objetos:

```
class MyClass:  
    x = 5  
  
p1 = MyClass()  
print(p1.x)
```

A terminal window with a black background and a white vertical bar on the left. The number 5 is displayed in white text.

5

# Desarrollo y utilización de clases y objetos.

## *La función `__init__()`*

Los ejemplos anteriores son clases y objetos en su forma más simple y no son realmente útiles en aplicaciones de la vida real.

Para entender el significado de las clases tenemos que entender la función incorporada `__init__()`.

Todas las clases tienen una función llamada `__init__()`, que siempre se ejecuta cuando se inicia la clase.

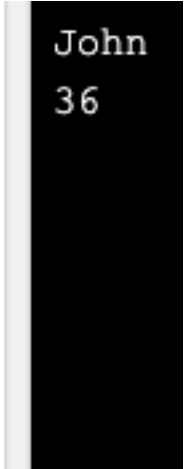
Utilice la función `__init__()` para asignar valores a las propiedades del objeto u otras operaciones que sean necesarias cuando se crea el objeto:

# Desarrollo y utilización de clases y objetos.

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

p1 = Person("John", 36)

print(p1.name)
print(p1.age)
```



John  
36

# Desarrollo y utilización de clases y objetos.

## Declaración

```
class Coche:  
    def __init__(self, gasolina):  
        self.__gasolina = gasolina  
        print("Tenemos", gasolina, "litros")
```

## Instanciación

```
mi_auto = Coche(5)
```

# Clases y objetos – Miembros de clase

Atributos de clase

```
class Coche:  
  
    marca = None #Atributo de clase público  
    __modelo = None #Atributo de clase privado
```

Método de clase

```
@classmethod  
def imprimir(self):  
    print("Yo soy un coche", self.marca)
```



# Clases y objetos – Miembros de clase

## Atributos de instancia

```
def __init__(self, gasolina, marca):  
    self.marca = marca #Atributo de instancia público  
    self.__gasolina = gasolina #Atributo de instancia privado  
    print("Tenemos", gasolina, "litros")
```

## Método de instancia

```
def arrancar(self):  
    if self.__gasolina > 0:  
        print("Arranca")  
    else:  
        print("No arranca")
```

# Clases y objetos – Miembros de clase

## *Métodos de objetos / instancias*

Los objetos también pueden contener métodos. Los métodos en los objetos son funciones que pertenecen al objeto.

Creemos un método en la clase Person:

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def myfunc(self):
        print("Hello my name is " + self.name)

p1 = Person("John", 36)
p1.myfunc()
```

**Nota:** El parámetro `self` es una referencia a la instancia actual de la clase y se usa para acceder a las variables que pertenecen a la clase.

```
Hello my name is John
```

# Clases y objetos – Miembros de clase

## El parámetro *self*

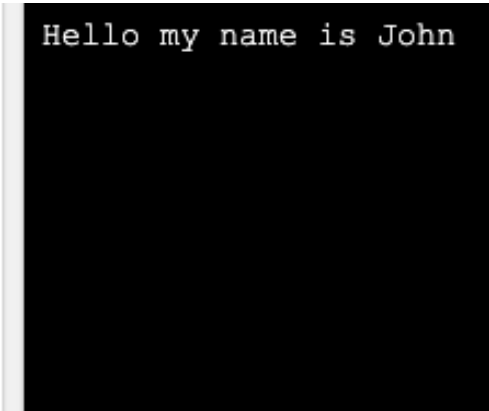
El parámetro *self* es una referencia a la instancia actual de la clase y se usa para acceder a las variables que pertenecen a la clase.

No tiene que tener un nombre *self*, puedes llamarlo como quieras, pero tiene que ser el primer parámetro de cualquier función en la clase:

```
class Person:
    def __init__(mysillyobject, name, age):
        mysillyobject.name = name
        mysillyobject.age = age

    def myfunc(abc):
        print("Hello my name is " + abc.name)

p1 = Person("John", 36)
p1.myfunc()
```

A terminal window with a black background and white text. The text "Hello my name is John" is displayed on a single line.

```
Hello my name is John
```

# Clases y objetos – Miembros de clase

## *Diferentes tipos de métodos en Python*

En Python existen 3 tipos de métodos. Los métodos **estáticos**, **de clase** y **de instancia**. Cada uno de ellos tienen características diferentes y se pueden utilizar en distintas situaciones.

### **Métodos Estáticos**

Para poder crear un método estático es necesario anteponer **@staticmethod** para así indicarle a Python que el método debe de ser estático. Las características principales de un método estático es que **pueden ser llamados sin tener una instancia de la clase**, además este tipo de métodos **no tienen acceso al exterior**, por lo cual no pueden acceder a ningún otro atributo o llamar a ninguna otra función dentro de la clase.

Un método estático puede ser utilizado cuando se tiene una clase pero no necesariamente se tiene una instancia para poder acceder al método. Por ejemplo si se tiene una clase **Math** y se tiene un método llamado **factorial** (calcula el factorial de un número dado).

Probablemente no se necesite tener una instancia específica para llamar al método, debido a esto se podría decidir hacer este método estático.

# Clases y objetos – Miembros de clase

```
1  class Math:
2      @staticmethod
3      def factorial(number):
4          if number == 0:
5              return 1
6          else:
7              return number * Math.factorial(number - 1)
8
9  factorial = Math.factorial(5)
10 print(factorial)
11
```

# Clases y objetos – Miembros de clase

## *Método de clase*

Para indicarle a Python que el método deberá ser un método de clase se debe de anteponer `@classmethod`, este método comparte una característica con el método estático, dicha característica es que este método puede ser llamado sin crear una instancia de la clase.

La diferencia recae en la capacidad de acceder otros métodos y atributos de la clase. Sin embargo este tipo de métodos no tienen accesos a atributos de instancia ya que ninguna instancia fue creada para poder utilizarlos

# Clases y objetos – Miembros de clase



## *Métodos de instancia*

Este método solamente puede ser llamado si se tiene una instancia de la clase.

Una vez que se creó una instancia de la clase, se podrá acceder a los métodos de instancia. Un método de instancia es capaz de crear, obtener y cambiar los atributos de instancia y a su vez de llamar otros métodos de instancia y clase

# Clases y objetos – Miembros de clase

```
1 class MethodTypes:
2
3     name = "Pedro"
4
5     def instanceMethod(self):
6         # Creates an instance attribute through keyword self
7         self.lastname = "Garcia"
8         self.__name2="Andrés"
9         print(self.name + " " + self.__name2)
10        print(self.lastname)
11
12    @classmethod
13    def classMethod(cls):
14        # Access a class attribute through keyword cls
15        cls.name = "Gustavo"
16        print(cls.name)
17
18    @staticmethod
19    def staticMethod():
20        print("Esto es un método estático")
21
22    # Creates an instance of the class
23    m = MethodTypes()
24    # Calls instance method
25    m.instanceMethod()
26
27    MethodTypes.classMethod()
28    MethodTypes.staticMethod()
29    m.instanceMethod()
```



C:\Program Files (x86)\Microsoft Visual Studio\Sh

```
Pedro Andrés
Garcia
Gustavo
Esto es un método estático
Gustavo Andrés
Garcia
Press any key to continue . . .
```



# Desarrollo y utilización de clases y objetos.

## *El parámetro self*

El parámetro `self` es una referencia a la instancia actual de la clase y se usa para acceder a las variables que pertenecen a la clase.

No tiene que tener un nombre `self`, puedes llamarlo como quieras, pero tiene que ser el primer parámetro de cualquier función en la clase.

```
class Person:
    def __init__(this, nombre, edad):
        this.nombre = nombre
        this.edad = edad

    def muestra_datos(abc):
        print("Mi nombre es " + abc.nombre + " y
tengo " + str(abc.edad) + " años")

p1 = Person("Juan", 43)
p1.muestra_datos()
```

Mi nombre es Juan y tengo 43 años

# Desarrollo y utilización de clases y objetos.

*Se pueden borrar atributos con **del**: del p1.edad*

```
1 class Person:
2     def __init__(this, nombre, edad):
3         this.nombre = nombre
4         this.edad = edad
5
6     def muestra_datos(abc):
7         print("Mi nombre es " + abc.nombre + " y tengo " + str(abc.edad) + " años")
8
9 p1 = Person("Juan", 43)
10 p1.muestra_datos()
11 del p1.edad
12 p1.muestra_datos()
13
```

Excepción producida

'Person' object has no attribute 'edad'

[Copiar detalles](#) | [Iniciar sesión de Live Share...](#)

# Desarrollo y utilización de clases y objetos.

*Se pueden borrar instancias con **del**: del p1.edad*

```
1 class Person:
2     def __init__(this, nombre, edad):
3         this.nombre = nombre
4         this.edad = edad
5
6     def muestra_datos(abc):
7         print("Mi nombre es " + abc.nombre + " y tengo " + str(abc.edad) + " años")
8
9 p1 = Person("Juan", 43)
10 p1.muestra_datos()
11 del p1
12 p1.muestra_datos()
13
```

Excepción producida

name 'p1' is not defined

[Copiar detalles](#) | [Iniciar sesión de Live Share...](#)

# Desarrollo y utilización de clases y objetos.

## Definición

```
def __init__(self, gasolina): #Constructor
    self.__gasolina = gasolina
    print("Tenemos", gasolina, "litros")

def __del__(self): #Destructor
    print("Eliminando objeto...")
```

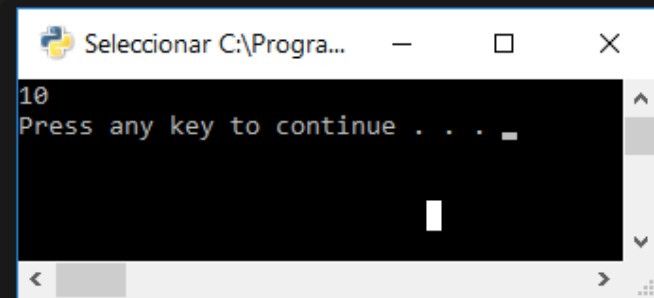
## Uso

```
auto2 = Coche(10)
del auto2
```

# Desarrollo y utilización de clases y objetos.

## *Encapsulamiento*

```
1 class Auto:
2     def GetNafta(self):
3         return self.__Nafta
4     def SetNafta(self,Litros):
5         if Litros>0 :
6             self.__Nafta = Litros
7         else:
8             print("No se puede ingresar un valor negativo !!!!")
9     Nafta = property(GetNafta,SetNafta)
10
11 A =Auto()
12 A.Nafta=10
13 print(A.Nafta)
```



# Desarrollo y utilización de clases y objetos.

Los miembros privados no son verdaderamente privados...

```
class Ejemplo:  
    def publico(self):  
        print ("Publico")  
    def __privado(self):  
        print ("Privado")  
  
ej = Ejemplo()  
ej.publico()  
ej._Ejemplo__privado()
```

# Ejemplo 1 - DEFINICIÓN DE CLASE

```
class Coche:

    marca = None #Atributo de clase público
    __modelo = None #Atributo de clase privado

    def __init__(self, gasolina): #Constructor
        self.__gasolina = gasolina
        print("Tenemos", gasolina, "litros")

    def __del__(self): #Destructor
        print("Eliminando objeto...")

    @classmethod
    def setModelo(self, modelo):
        self.__modelo = modelo
```

# Ejemplo 1 - DEFINICIÓN DE CLASE

```
def getGasolina(self):  
    return self.__gasolina  
  
def setGasolina(self, gasolina):  
    if gasolina > 0:  
        self.__gasolina = gasolina  
    else:  
        print("No se puede cargar menos o igual a 0 de combustible")  
  
gasolina = property(getGasolina, setGasolina)  
  
def arrancar(self):  
    if self.__gasolina > 0:  
        print("Arranca")  
    else:  
        print("No arranca")  
  
def conducir(self):  
    if self.__gasolina > 0:  
        self.__gasolina -= 1  
        print("Quedan", self.__gasolina, "litros")  
    else:  
        print("No se mueve")
```



# Ejemplo 1 - DEFINICIÓN DE CLASE

```
@classmethod
def imprimir(self):
    print("Yo soy un coche", self.marca, "modelo", self.__modelo)

@staticmethod
def funcion_sin_relacion_con_la_clase(origen, destino, kms): #Definimos el metodo
    print ("El automovil irá desde", origen, "hasta", destino, "recorriendo", kms, "kms")
```

# Ejemplo 1 - USO

```
auto1 = Coche(5) #Instancio un coche
Coche.marca = "Fiat" #Modifico atributo público de clase
Coche.setModelo("Argo") #Modifico atributo privado de clase (setter)
auto1.arrancar() #Arrancar el vehículo, método de instancia
auto1.conducir() #Conducir el vehículo, método de instancia
#auto1.setGasolina(10)
auto1.setGasolina(0) #Método setter tradicional
auto1.gasolina = 10 #Seteo por property
print(auto1.gasolina) #Traigo la gasolina por property
print(auto1.getGasolina()) #Traigo la gasolina por getter tradicional
auto1.conducir()
auto1.conducir()
Coche.imprimir() #Método de clase -> LLamo directo a la clase
Coche.funcion_sin_relacion_con_la_clase("bs as", "rosario", 45) #Método estático -> Sin relación interna
del auto1 #Destruyo el objeto
```

# Ejemplo 2 - Composición

```
#Composición
class Salary:
    def __init__(self, pay):
        self.__pay=pay

    def get_total(self):
        return (self.__pay * 12)

class Employee:
    def __init__(self, pay, bonus):
        self.__pay=pay
        self.__bonus=bonus
        self.__obj_salary=Salary(self.__pay) #Compongo a la clase Salarario

    def annual_salary(self):
        return "Total: " + str(self.__obj_salary.get_total() + self.__bonus)

obj_emp=Employee(100, 10)
print (obj_emp.annual_salary())
```

# Ejemplo 3 - Agregación

```
#Agregación
class Salary:
    def __init__(self, pay):
        self.__pay=pay

    def get_total(self):
        return (self.__pay * 12)

class Employee:
    def __init__(self, pay, bonus):
        self.__pay=pay #Paso directamente el objeto Salario en el constructor
        self.__bonus=bonus
        #self.__obj_salary=Salary(self.__pay)

    def annual_salary(self):
        return "Total: " + str(self.__pay.get_total() + self.__bonus)

obj_salary=Salary(100) #Creo el objeto salario
obj_emp=Employee(obj_salary, 10) #Lo paso como parámetro al constructor a Empleado
print (obj_emp.annual_salary())
```

# Herencia

## *Herencia en Python*


La herencia nos permite definir una clase que hereda todos los métodos y propiedades de otra clase.

**La clase principal** es la clase de la que se hereda, también llamada clase base.

**La clase hija** es la clase que hereda de otra clase, también llamada clase derivada.

# Herencia

```
1  # Clase principal / class base
2  class Persona:
3      def __init__(self, fname, lname):
4          self.firstname = fname
5          self.lastname = lname
6
7      def printname(self):
8          print("Hola mi nombre es: " + self.firstname, self.lastname)
9
10 class Alumno(Persona):
11     def printname(self):
12         print("Hola soy un alumno y mi nombre es: " + self.firstname, self.lastname)
13
14
15 x = Persona("John", "Doe")
16 x.printname()
17 print()
18 print()
19
20 y = Alumno("Ana", "Garcia")
21 y.printname()
22 print()
23 print()
```

 C:\Program Files (x86)\Microsoft Visual Studio\Shared\Python37\_64\python.exe

Hola mi nombre es: John Doe


Hola soy un alumno y mi nombre es: Ana Garcia

Press any key to continue . . .

# Herencia

## *Agregando un constructor a Alumno*

```
1 class Persona:
2     def __init__(self, fname, lname):
3         self.firstname = fname
4         self.lastname = lname
5
6     def printname(self):
7         print("Hola mi nombre es: " + self.firstname, self.lastname)
8
9 class Alumno(Persona):
10     def __init__(self, fname, lname):
11         Persona.__init__(self, fname, lname)
12
13     def printname(self):
14         print("Hola soy un alumno y mi nombre es: " + self.firstname, self.lastname)
15
16 x = Persona("John", "Doe")
17 x.printname()
18 print()
19 print()
20
21 y = Alumno("Ana", "Garcia")
22 y.printname()
23 print()
24 print()
```

 C:\Program Files (x86)\Microsoft Visual Studio\Shared\Python37\_64\python.exe

Hola mi nombre es: John Doe

Hola soy un alumno y mi nombre es: Ana Garcia


Press any key to continue . . .

# Herencia

## *Agregando un constructor a Alumno*

```
1 class Persona:
2     def __init__(self, fname, lname):
3         self.firstname = fname
4         self.lastname = lname
5
6     def printname(self):
7         print("Hola mi nombre es: " + self.firstname, self.lastname)
8
9 class Alumno(Persona):
10     def __init__(self, fname, lname):
11         super().__init__(fname, lname)
12
13     def printname(self):
14         print("Hola soy un alumno y mi nombre es: " + self.firstname, self.lastname)
15
16 x = Persona("John", "Doe")
17 x.printname()
18 print()
19 print()
20
21 y = Alumno("Ana", "Garcia")
22 y.printname()
23 print()
24 print()
```

Al usar la función `super()`, no tiene que usar el nombre del elemento padre, automáticamente heredará los métodos y propiedades de su padre.

 C:\Program Files (x86)\Microsoft Visual Studio\Shared\Python37\_64\python.exe

Hola mi nombre es: John Doe

Hola soy un alumno y mi nombre es: Ana Garcia

Press any key to continue . . .



# Ejercicio

Desarrollar un programa que permita administrar las cajas de un supermercado.

Cada caja posee: una fila de personas que desean abonar sus compras y otra con las que ya han abonado.

La caja posee la posibilidad de cobrarle la compra al cliente. Un cliente además de sus características propias posee un atributo que representa la cantidad total de pesos que debe abonar.

El dueño del supermercado desea conocer:

1. Cuántos clientes hay en cada caja que aún no han pagado.
2. Cuánto dinero hay en cada caja por cobrar.
3. Cuántos clientes hay en cada caja que ya han pagado.
4. Cuánto dinero se ha cobrado en cada caja.
5. El total de clientes por cobrarles considerando todas las cajas.
6. El total del dinero que resta cobrar considerando todas las cajas.
7. El total de clientes cobrados considerando todas las cajas.
8. El total del dinero cobrado considerando todas las cajas.

# Ejercicio



## Clases:

Supermercado

Caja

Cliente

## Relaciones:

Supermercado posee \* cajas (Composición)

Caja posee \* clientes que deben abonar su compra (Composición)

Caja posee \* clientes que ya abonaron su compra (Composición)

# Ejercicio

## Supermercado

Atributos:

Nombre

Acciones:

1. El total de clientes por cobrarles considerando todas las cajas.
2. El total del dinero que resta cobrar considerando todas las cajas.
3. El total de clientes cobrados considerando todas las cajas.
4. El total del dinero cobrado considerando todas las cajas.
5. Seleccionar la caja para encolar al Cliente

# Ejercicio

Caja

Atributos:

Número

Acciones:

1. Cuántos clientes hay en cada caja que aún no han pagado.
2. Cuánto dinero hay en cada caja por cobrar.
3. Cuántos clientes hay en cada caja que ya han pagado.
4. Cuánto dinero se ha cobrado en cada caja.
5. Encolar Cliente.
6. Cobrar.

# Ejercicio



Cliente

Atributos:

Nombre

Monto

Acciones:

# Ejercicio

Bienvenido a ....

Cuántas cajas posee el supermercado?

\*\*\*\*\* Menu \*\*\*\*\*

1. Encolar Cliente
2. Cobrar
3. Salir

Q Clientes por Cobrar C1:  
Q Clientes Cobrados C1:  
\$ Clientes por Cobrar C1:  
\$ Clientes Cobrados C1:  
Por Cobrar: Juan, Pedro, Ana  
Cobrados: Sol, Analís, Ferando

\*\*\*\*\*

Q Clientes por Cobrar C2:  
Q Clientes Cobrados C2:  
\$ Clientes por Cobrar C2:  
\$ Clientes Cobrados C2:  
Por Cobrar: Nicolás, Aldo, Jimena  
Cobrados: Soledad, Patricia, Cecilia

\*\*\*\*\*

\*\*\*\*\* Totales \*\*\*\*\*

Q Clientes por Cobrar:  
Q Clientes Cobrados :  
\$ Clientes por Cobrar :  
\$ Clientes Cobrados :